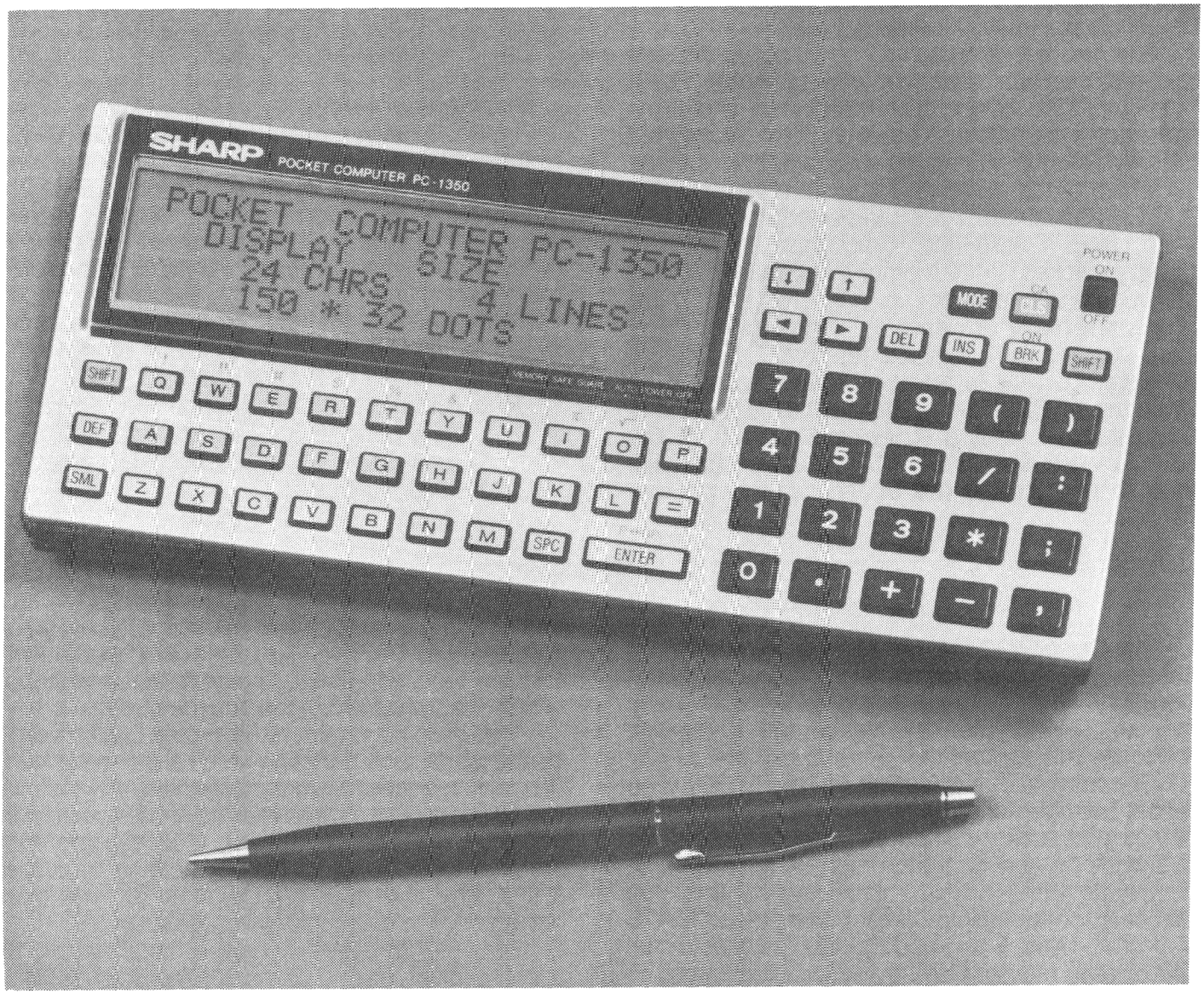
POCKET COMPUTER NEWSLETTER

© Copyright 1984 POCKET COMPUTER NEWSLETTER

Issue 36 - November/December

Photo *Sharp PC-1350 Pocket Computer Sports Multiple-Line Liquid-Crystal Display.*



Another *Personal Information* ™ product.

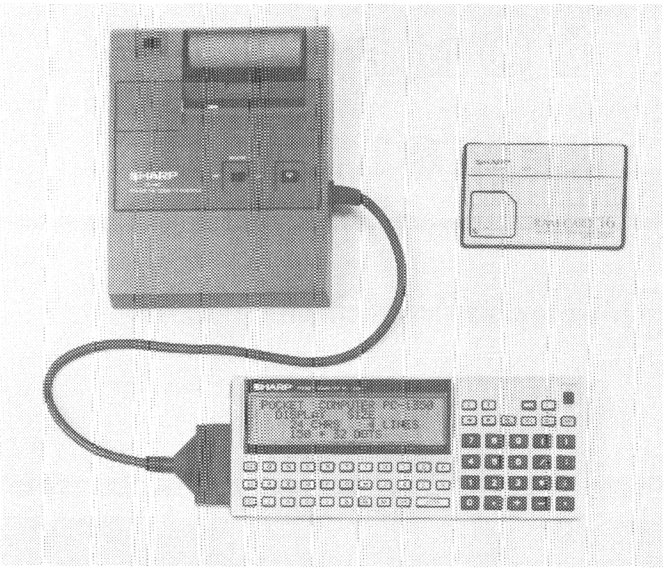
SHARP PRODUCING NEW MODEL PC-1350

Sharp Electronics Corporation is now delivering a new, slim pocket computer dubbed the model PC-1350. It is believed that the model may eventually replace the popular PC-1500 produced by the company. The unit weighs less than seven ounces. It is just slightly over 1/2 inch in thickness, with a length of approximately 7-1/4 inches and a width under 3 inches. The PC-1350 sports a large 4 line by 24 characters display. The U.S. list price is said to be $195.00 at the present time.

The unit's keyboard contains 62 keys. Thirty-two of these are arranged in QWERTY fashion. Twenty are grouped in a numeric/operand keypad. Ten more serve as cursor, scrolling, mode and editing aids. There is a separate power on/off switch. A contrast control, accessed on the right-hand side of the unit, provides for adjustment of the liquid-crystal display intensity.

The new pocket machine has a power-packed software package residing in 40 kilobytes of ROM. This includes a BASIC interpreter and user operating system. The PC also contains 5 kilobytes of user RAM provided as standard. This can be expanded by an additional 8 or 16 kilobytes. The RAM expansion units are somewhat unusual. They are thin, card-like modules containing battery-backed CMOS memory. Each card can hold programs and/or data for up to several years. There is only one major drawback to using these cards as mass storage devices: their relative cost. The 8K RAM card is said to retail at approximately $100 while the 16K version will be about $170.00. That is equivalent to what a RAM module such as the CE-161 costs on the PC-1500. Such a price will tend to restrict the use of the cards as general purpose storage devices.

Photo *PC-1350 with CE-126P Printer/Cassette Unit and CE-202M (16K) Memory Card.*



The PC-1350 is powered by a pair of CR-2032 lithium cells. It is equipped with an interface enabling it to be coupled to a CE-126P printer/cassette interface. A second serial interface provides a means of interchanging information with Sharp's PC-5000 Portable Computer.

Will the new PC-1350 become the volume pocket computer leader? Will Radio Shack soon begin promoting a PC-5? Only time will tell.

To obtain more information on the PC-1350, contact **Sharp Electronics Corporation,** *10 Sharp Plaza, Paramus, NJ 07652.* The telephone number is (201) 265-5600.

REVIEW OF THE CASIO PB-700

The Casio PB-700 is the same size as the Radio Shack PC-2 Pocket Computer. It has a staggered QWERTY keyboard. The keyboard includes a key marked "CAPS". This key may be used to provide lower case characters.

The most striking feature of the PB-700 upon initial examination is its display. It is the largest of any current pocket computer. It measures 4 inches in length by almost 1 inch high. It is capable of displaying 4 lines with 20 characters per line. You can position the display cursor using the LOCATE X,Y statement. When used in the graphics mode, the display provides 160 by 32 dots (compared to the PC-2's 160 by 7 dots). It is easy to draw pictures on the display. Just use the statement DRAW X,Y. To erase, use DRAWC X,Y. The POINT statement can be used to have a program automatically determine the state of a pixel on the screen.

There are no annunciators on the LCD screen. To ascertain such things as which angle mode is in use, the amount of memory remaining, or which of the 10 possible program areas have been used, you utilize a SYSTEM command. You execute the program in a particular area by holding down the SHIFT key and pressing the digit (0 - 9) that corresponds to the program storage area.

The PB-700 does not have a mode switch. It does, however, have a connector that may be used to attach a printer or cassette interface, both of

which are available as options. It also has a memory bay on the back that can hold up to 3 RAM modules. Each RAM module can contain 4K of memory. The RAM modules currently retail for $39.95 each. Since the basic PB-700 includes 4K of user memory, a fully-loaded unit can have up to 16 kilobytes of RAM. However, about 2.8K of RAM is consumed as overhead.

The unit is powered by 4 type AA batteries. They are accessed via a door in the back of the machine.

The BASIC instruction set of the PB-700 includes the following directives: POINT, NEW, LIST, CONT, RUN, BEEP, CHAIN, CLEAR, CLS, DATA, DIM, END, GOSUB, RETURN, GOTO, FOR, TO, STEP, NEXT, INPUT, LET, PRINT, LPRINT, READ, REM, RESTORE, STOP, TRON and TROFF. It provides a number of string functions as well as trigonometric and inverse-trig functions. It has EXP, SQR, LOG (natural) and LGT (base 10 log), ABS, INT, SGN, RND and ROUND functions as well as INKEY$, USING and TAB. The ROUND(X,Y) function will round X to the decimal position indicated by Y. Thus, for instance, if Y is 2, the value X will be rounded to the nearest hundredths place!

When compared with the Radio Shack PC-2, the Casio PB-700 lacks the following statements and commands: TIME, PAUSE, CALL, PEEK, POKE, ARUN, AREAD, BEEP ON, ON ERROR GOTO, NOT, STATUS, WAIT, LOCK and UNLOCK.

However, it does have features that are not found on the PC-2, such as:

1. The ability to use non-integer values in FOR, NEXT and STEP statements.

2. More flexibility within IF-THEN-ELSE statements.

3. The use of half-precision array variables (5-digit mantissa with a 2- digit exponent) instead of full-precision variables (12-digit mantissa and 2- digit exponent). This can save a lot of memory space when manipulating arrays.

4. A key that makes it easy to delete a lot of characters. Once the cursor has been positioned, one press of this key will delete everything to the right on a particular line.

The PB-700 permits the use of 1- and 2-dimensional arrays as well as 2- character array variable names.

The unit is supplied with a carrying case, a manual, a quick reference booklet, 4 AA batteries and a RAM backup battery. The manual contains demonstration programs including one that provides impressive graphics.

The PB-700 has a U.S. list price of $199.95. It is available through Casio distributors.

This review provided by *Jeffrey Woodhead, P.O. Box 770, Davis, CA 95617.*

FOR PC-1250/51/60/61 USERS

ALARM CLOCK PROGRAM

This program converts a Sharp PC into a "smart" alarm clock. Several versions of the program are provided: one for PC-1250/1251s (or the equivalent Radio Shack PC-3) and another for PC-1260/1261 owners. Furthermore, a slight modification to the version for the 1250 will enable it to be used on the original Sharp PC-1211 (Radio Shack PC-1)! The program can be "calibrated" so that it is accurate to within a few seconds per day. It is "smart" in that it allows the SET and ALARM time to occur at any time within a 24 hour period. Thus, for example, you can set it at 8:00AM for an alarm time of 7:00AM and it will "know" that you mean the next morning. It won't bother you at 7:00PM as a 12-hour clock would!

The input times are entered in the format (HH.MMSS) followed by AM or PM (when using the PC-1250/1251 version). When using the 1260/1261 version, you should follow the time by the notation *A*M or *P*M. (The asterisk meaning multiplication. The reason that the multiplication symbol must be used in the 1260/1261 version is because it does not recognize "implied" multiplication. The symbols A, P, and M are actually used in the program as variable names. They assume values of plus or minus one. These types of differences amongst supposedly compatible PCs stems from the fact that the manufacturer assumes that you follow rigid programming conventions. The use of "implied" multiplication as done in the 1250/1251 version is not sanctioned by Sharp.)

In any event, if you do not specify AM (*A*M) or PM (*P*M) when inputting a time value, the program will assume you mean AM. Also, the program assumes that noon is 12.00AM and midnite is 12.00PM. The program is sufficiently accurate that you may wish to

Program *Alarm Clock for PC-1250/51 (Or PC-3).*

```
10:BEEP 1: PAUSE " 24 H
   R ALARM CLOCK"
12:REM "NOON = 12.00AM"
14:REM "MIDNITE = 12.00
   PM"
20:Y=3.225597096E-4:A=1
   :P=-1:M=1: USING "##
   #.####"
30:INPUT "ALARM TIME(HH
   .MM):";U
40:INPUT "TIME NOW(HH.M
   M):";S:Z= DEG ABS S
50:IF U>0 IF ABS U>12
   LET U=U-12
55:IF S>0 IF ABS S>12
   LET S=S-12
60:IF U<0 LET U= ABS U+
   12: IF U>24 LET U=U-
   12
65:IF S<0 LET S= ABS S+
   12: IF S>24 LET S=S-
   12
70:V= DEG U- DEG S: IF
   V<0 LET V=V+24
80:V=V+Z:Z=Z+0.7250Y:X=
   Z
90:IF X>=V THEN 130
100:Z=Z+Y: IF Z>=13 LET
    Z=Z-12
110:X=X+Y:T= DMS Z
120:PAUSE "TIME:";T:
    GOTO 90
130:USING "###.##"
140:BEEP 3: PAUSE "TIME:
    ";T
150:Z=Z+1.55782Y: IF Z>=
    13 LET Z=Z-12
160:T= DMS Z: GOTO 140
200:"C": PAUSE "CALIBRAT
    ION -->": USING
210:INPUT "TRUE CLOCK(H.
    MMSS):";W
220:IF W>0 IF ABS W>12
    LET W=W-12
230:IF W<0 LET W= ABS W+
    12: IF W>24 LET W=W-
    12
240:J= DEG U- DEG S: IF
    J<0 LET J=J+24
250:K= DEG W- DEG S: IF
    K<0 LET K=K+24
260:L=KY/J: PRINT "NEW Y
     = ";L: END
```

Program *Alarm Clock for PC-1260/1261.*

```
10:BEEP 1: PAUSE " 24 H
   R ALARM CLOCK"
12:REM "NOON = 12.00AM"
14:REM "MIDNITE = 12.00
   PM"
20:Y=3.251197073E-4:A=1
   :P=-1:M=1: USING "##
   #.####"
30:INPUT "ALARM TIME(HH
   .MM):";U
40:INPUT "TIME NOW(HH.M
   M):";S:Z= DEG ABS S
50:IF U>0 IF ABS U>12
   LET U=U-12
55:IF S>0 IF ABS S>12
   LET S=S-12
60:IF U<0 LET U= ABS U+
   12: IF U>24 LET U=U-
   12
65:IF S<0 LET S= ABS S+
   12: IF S>24 LET S=S-
   12
70:V= DEG U- DEG S: IF
   V<0 LET V=V+24
80:V=V+Z:Z=Z+0.7250*Y:X
   =Z
90:IF X>=V THEN 130
100:Z=Z+Y: IF Z>=13 LET
    Z=Z-12
110:X=X+Y:T= DMS Z
120:PAUSE "TIME:";T:
    GOTO 90
130:USING "###.##"
140:BEEP 3: PAUSE "TIME:
    ";T
150:Z=Z+1.55782*Y: IF Z>
    =13 LET Z=Z-12
160:T= DMS Z: GOTO 140
200:"C": PAUSE "CALIBRAT
    ION -->": USING
210:INPUT "TRUE CLOCK(H.
    MMSS):";W
220:IF W>0 IF ABS W>12
    LET W=W-12
230:IF W<0 LET W= ABS W+
    12: IF W>24 LET W=W-
    12
240:J= DEG U- DEG S: IF
    J<0 LET J=J+24
250:K= DEG W- DEG S: IF
    K<0 LET K=K+24
260:L=K*Y/J: PRINT "NEW
    Y = ";L: END
```

enter times to the nearest second. This is accomplished by appending the seconds value after minutes, i.e., entering a time as 12.5315PM (12.5315*P*M on a 1260/1261).

Suppose you want to have the PC tell you when it is 7:15 in the morning? Assume that the current time is 10:35PM. You would run the program as follows:

```
>RUN
24 HOUR ALARM CLOCK
ALARM TIME(HH.MM):  7.15*A*M
TIME  NOW (HH.MM)  10.35*P*M
.... at 7.15AM the next morning ....
BEEP  -  BEEP  -  BEEP
TIME:  7.15
```

Suppose you found the alarm clock program to be slightly off as far as timing was concerned? You can calibrate it against a chronometer (er, perhaps your trusty wrist watch will suffice) having a second hand, if you desire. Let's assume that you ran the program as in the example above, but the alarm went off at an actual time (according to your chronometer) of exactly 7:17:30 in the AM. To enhance the accuracy of the program for your specific PC, you do the following:

```
>RUN 200   or  DEF/C
CALIBRATION -->
TRUE CLOCK(H.MMSS): 7.1730*A*M
NEW Y = WHATEVER IT SAYS
```

You complete the adjustment by placing the PC into the PROGRAM mode and changing the calibration constant Y in line 20 to the NEW Y value given!

To use the program on an original PC-1211 (Radio Shack PC-1) you can use the program listing given for the PC-1250/1251 but start with a value for Y in line 20 of Y=5.561374309E-04.

It is a pretty nifty little program. Thanks for it go to: *Emerich Auersbacher, 41 King Street #2, Belleville, NJ 07109.*

FOR HEWLETT-PACKARD HP-71B USERS

WEATHER PROGRAM FOR HP-71B

Ever since the introductory announcement in Issue 32 of *PCN* and the subsequent product review article in Issue 33, we have been getting more and more requests for additional information and programs specifically for the HP-71B. OK, OK, we get the message!

It sure looks like HP has cooked up a winner with the 71B. We haven't had as many requests to cover a particular model since Sharp Electronics Company introduced the PC-1500.

So, to kick things off in what we plan to be continuing coverage of this model, we offer up a HP-71B-tailored weather forecasting program. Long-time *PCN* readers may recognize this as a customized version of *E. Auersbacher's* program originally published in Issue 20 for the Radio Shack PC-2 and Sharp PC-1500. (By the way, line 500 of the original version contained a typo - the IF Z=3 statement should have been IF X=3!) Operation of the program is straightforward. So, just load and RUN. We will have more for the 71B next issue!

Program *Weather Forecasting on the HP-71B. (Listing continued on the next page.)*

```
10 DELAY 1,.5 @ WIDTH 96 @ DISP "WEATHER FORECASTER @ DESTROY ALL
20 A$="S. SW W. NW N. NE E. SE "
30 INPUT "BAROMETER (IN): ";P @ IF P>=30.2 THEN S=0
40 IF P<30.2 THEN S=1
50 IF P<30.1 THEN S=2
60 IF P<30 THEN S=3
70 IF P<=29.8 THEN S=4
80 INPUT "RISE, FALL, STEADY? ";T$ @ X=0
90 IF FNA$(T$)#"R" THEN 130
```

Program *Weather Forecasting on the HP-71B. (Listing starts on previous page.)*

```
100 INPUT "RISING FAST, SLOW? ";R$ @ IF FNA$(R$)="F" THEN X=1 @ GOTO 190
110 IF FNA$(R$)#"S" THEN BEEP @ GOTO 100
120 X=2 @ GOTO 190
130 IF FNA$(T$)="S" THEN 190
140 IF FNA$(T$)#"F" THEN BEEP @ GOTO 80
150 INPUT "FALLING FAST, SLOW? ";R$
160 IF FNA$(R$)="F" THEN X=3 @ GOTO 190
170 IF FNA$(R$)#"S" THEN BEEP @ GOTO 140
180 X=4
190 INPUT "WIND FROM THE: ";W$ @ IF LEN(W$)#2 THEN BEEP @ GOTO 190
200 W=POS(A$,UPRC$(W$)) @ IF W=0 THEN BEEP @ GOTO 190
210 W=INT(W/3)+1
300 Y=ABS(W-3)<2
310 IF Y*(S<3) AND X=0 THEN Z=1 @ GOTO 600
320 IF Y*(S=1) AND X=1 THEN Z=6 @ GOTO 600
330 IF Y=(S=0) AND X=4 THEN Z=3 @ GOTO 600
340 Y=W=1 OR W=8
350 IF Y*(S=1) AND X=4 THEN Z=4 @ GOTO 600
360 IF Y*(S=1) AND X=3 THEN Z=5 @ GOTO 600
370 Y=ABS(W-7)<2
380 IF Y*(S=1) AND X=4 THEN Z=4 @ GOTO 600
390 IF Y*(S=1) AND X=3 THEN Z=5 @ GOTO 600
400 IF Y*(S>2) AND X=4 THEN Z=6 @ GOTO 600
410 IF Y*(S>2) AND X=3 THEN Z=7 @ GOTO 600
420 Y=W=6 OR W=7
430 IF Y*(S<2) AND X=4 THEN Z=8 @ GOTO 600
440 IF Y*(S<2) AND X=3 THEN Z=9 @ GOTO 600
450 IF (W=1 OR W=2) AND S>2 AND X=2 THEN Z=10 @ GOTO 600
460 IF (W<2 OR W>6) AND S=4 AND X=3 THEN Z=11 @ GOTO 600
470 IF ABS(W-6)<2 AND S=4 AND X=3 THEN Z=12 @ GOTO 600
480 IF ABS(W-3)<2 AND S=4 AND X=1 THEN Z=13 @ GOTO 600
490 iF X<3 AND ABS(W-5)<3 THEN Z=2 @ GOTO 600
500 Z=1 @ IF X=3 THEN Z=8
600 DISP "FORECAST -->" @ K$=""
610 ON Z GOSUB 710,720,730,740,750,760,770,780,790,800,810,820,830
620 K$=KEY$ @ IF K$="" THEN 610 ELSE 30
710 DISP "FAIR, LITTLE CHANGE." @ RETURN
720 DISP "FAIR, LITTLE COOLER." @ RETURN
730 DISP "FAIR, LITTLE WARMER." @ RETURN
740 DISP "RAIN WITHIN 24 HRS." @ RETURN
750 DISP "WINDY, RAIN BY 12 HRS." @ RETURN
760 DISP "CLOUDING WITH RAIN." @ RETURN
770 DISP "RAIN AND HIGH WINDS." @ RETURN
780 DISP "UNSETTLED, MAYBE RAIN." @ RETURN
790 DISP "RAIN/SNOW, WINDY." @ RETURN
800 DISP "CLEARING TO FAIR." @ RETURN
810 DISP "SEVERE STORM IMMINENT." @ RETURN
820 DISP "HEAVY RAIN/SNOW, WIND." @ RETURN
830 DISP "CLEARING AND COOLER." @ RETURN
900 DEF FNA$(R$)=UPRC$(R$[1,1])
```

FOR PC-1500 & PC-2 USERS

MACHINE LANGUAGE PROGRAMMING

Readers of the series *Machine Language Programming the Sharp PC-1500 and Radio Shack PC-2* (produced in 1983 as a separate publication by *PCN*) will especially appreciate the following article. As such readers know, the series was abruptly terminated due to serious illness on the part of the author.

What follows is a sample of how the author had planned to continue the series by building upon the instruction set foundation that had been laid in the early installments. Enjoy!

MACHINE LANGUAGE PROGRAMMING
THE SHARP PC-1500
AND RADIO SHACK PC-2
POCKET COMPUTERS

Once a truly interested ML fan acquires an understanding of the types of instructions that are available on a machine, he or she soon develops an itch to do some real ML programming. Let's satisfy that urge right now.

Clearing Memory

When batteries are installed in a PC, the individual bits in memory will "come up" in random states. Some set to the logic 1 state, some cleared to the 0 state. Sometimes it is not desirable to have areas of RAM in such a chaotic condition. One way for a ML programmer to set an area in memory to a known conditon is to fill it with zero bytes. That is, load bytes set to the value zero into a specific range of memory addresses. Sounds like a pretty simple procedure, right? It is! And it can be done in a whole lot of different ways.

Suppose, for example, that you wanted to clear out the section of memory normally used to store the fixed string variables A$ through D$. These are stored in memory addresses &78C0 - &78FF.

One way to accomplish this job would be to load CPU register A with the value zero. Next, the starting address of the area to be cleared might be set up in CPU register X. Once this had been done, STAI (X) directives could be used to stuff the contents of the accumulator (register A) into successive locations in memory as pointed to by the contents of CPU register X. Remember, the STAI (X) instruction automatically advances the value in X each time it is executed. There is just one more parameter to consider. How many times must the STAI (X) directive be repeated in order to clear out the desired block of memory?

Since the size of the memory block is known in this example, a counter could be established, say in register UL. Each time the STAI (X) command was performed, the count in UL could be decremented. When this count reached zero, it would be time to stop stuffing zeros into memory. Thus, one could use the sequence of directives (after the STAI (X) instruction) consisting of: DEUL and RBNZ #nn. That is, decrease the count in register UL and if it is still non-zero, then loop back to repeat the STAI (X) directive. If this method was used, in this example, then register UL would initially have to be set to the decimal count of 64 (hexadecimal 40), representing the number of bytes in memory (from &78C0 through &78FF) that were to be cleared. See the accompanying listing for a detailed example of this method.

If you were wide awake when you read the previous section of this series, then you might remember that this type of situation is ideal for the use of the BNZD #nn instruction. Since, however, the BNZD directive tests for zero *before* it decrements the contents in UL, then the count initially placed in UL must be one less than the number of loops (locations to be cleared). Thus, in this specific example, if BNZD was used, register UL would need to start out with a count of decimal 63 (which is hexadecimal &3F). An accompanying listing illustrates this method, too.

Yet another way of determining when to stop looping would be to check for an appropriate address value in CPU register X. In this case, when the value in X exceeded &78FF (i.e., reached &7900), then it would be time to discontinue the clearing operation. There are several ways this type of procedure might be implemented.

One way would be to set the ending value (say &78FF in this case) into another CPU register. CPU register Y would be available for such use in this example. A comparison could then be made between the contents of X and Y each time that X was advanced by the STAI directive. When the value in X exceeded that in Y, then it would be appropriate to stop the clearing operation. See the

Program *Routine for Clearing Memory Using RBNZ Instruction.*

PG	LC	B1	B2	B3	B4	B5	LABELS	MNEMONICS	COMMENTS
00	00	B5	00				CLRMM1	LDA #00	Load register A with zero.
00	02	48	78					LDXH #78	Set up register X to point to 16-bit address 678C0.
00	04	4A	C0					LDXL #C0	Set up register X to point to 16-bit address 678C0.
00	06	6A	40					LDUL #40	Set up counter (64 decimal here) in register UL.
00	08	41					CLRM1	STAI (X)	Stuff contents of A into memory, then increment pointer.
00	09	62						DEUL	Decrement the counter value in UL (UL=UL-1).
00	0A	99	04					RBNZ CLRM1	If counter is not zero, jump back to stuff another byte.

Program *Routine for Clearing Memory Using BNZD Instruction.*

PG	LC	B1	B2	B3	B4	B5	LABELS	MNEMONICS	COMMENTS
00	00	B5	00				CLRMM2	LDA #00	Load register A with zero.
00	02	48	78					LDXH #78	Set up register X to point to 16-bit address 678C0.
00	04	4A	C0					LDXL #C0	Set up register X to point to 16-bit address 678C0.
00	06	6A	3F					LDUL #3F	Set up counter (63 decimal here) to count-1 (64-1=63).
00	08	41					CLRM2	STAI (X)	Stuff zero byte into memory, advance memory pointer in X.
00	09	88	03					BNZD CLRM2	Check UL for zero, loop back if non-zero, (UL=UL-1).

Program *Routine for Clearing Memory Using Compare Operation to Terminate Loop.*

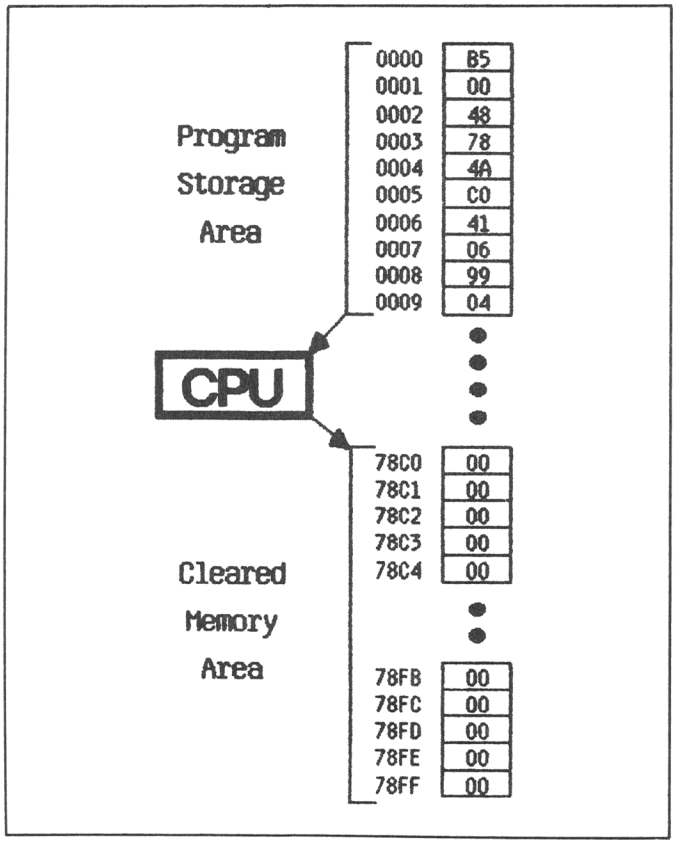
PG	LC	B1	B2	B3	B4	B5	LABELS	MNEMONICS	COMMENTS
00	00	B5	00				CLRMM3	LDA #00	Load register A with zero.
00	02	48	78					LDXH #78	Set up register X to point to 16-bit address 678C0.
00	04	4A	C0					LDXL #C0	Set up register X to point to 16-bit address 678C0.
00	06	41					CLRM3	STAI (X)	Stuff zero byte into memory, advance memory pointer in X.
00	07	06						CPA XL	Check XL for zero (indicating address 67900 reached).
00	08	99	04					RBNZ CLRM3	Loop back to stuff next location if XL is not zero.

example program listing.

An even easier way, in this particular case, would merely be to test for the value in XL going to zero. That is because, when XL exceeds FF (because X reaches 78FF), it will go to the value 00 as X advances to the value 7900. That just happens to be the point at which we want to stop clearing memory in this particular example! A listing of this method is also provided for examination.

By now you should be convinced that ML programming is not an exact science. There are usually countless ways to approach a particular objective. Sometimes you can customize your approach depending on specific parameters. For instance, if you are trying to conserve your use of memory, you might try to devise a sequence of directives that uses a minimum amount of space. Or, you might be interested in maximizing speed of program execution. In that case, you could work towards selecting directives that could be performed in the minimum amount of time. (Do *not* make the assumption that the fastest program is the one with the fewest instructions! Various classes of instructions require different times to complete their execution. It may require a very detailed study in order to find a sequence of instructions that accomplishes an objective in a minimum amount of time!) In most practical applications, however, the actual sequence of instructions selected is not at all critical. Select or create a method you like and try it out!

Diagram *Memory Map of Clearing Operation.*



A Practical Memory Clearing Application

Have you ever developed a BASIC program that used a temporary variable array? That is, an array that you needed to continuously clear out in between operations with other arrays? If so, you know that you had to create a special program loop that would initialize all the elements in that specific temporary array. You could not use a BASIC command such as CLEAR as that would wipe out *all* of the other variables and array elements used in the program, something that we assume for this example, would not be desired.

Of course, there is nothing wrong with creating a BASIC program loop to clear out the elements in an array. It is just that, if the number of elements in the array is large, the process can take some time. If the clearing operation has to be done frequently, the amount of time devoted to this one aspect can become quite exasperating.

However, as a ML programmer, it is possible to devise a scheme to clear out the elements of an array in the proverbial "blink of an eye." Let us see how this could be done.

First, it is necessary to know a few facts about the operation of the BASIC interpreter provided in the PC-1500 (and Radio Shack PC-2). As you may be aware, the interpreter program stored on ROM organizes RAM memory in a specific fashion to serve its purposes. Thus, the lower range of available user memory is assigned for use by the PC's "softkeys" as the REServe memory area. Immediately above this area (in terms of memory addresses) is where user program statements for a BASIC program (or multiple programs) are stored. Finally, the "top" or highest address value locations in RAM are used for the storage of user-defined arrays and variables.

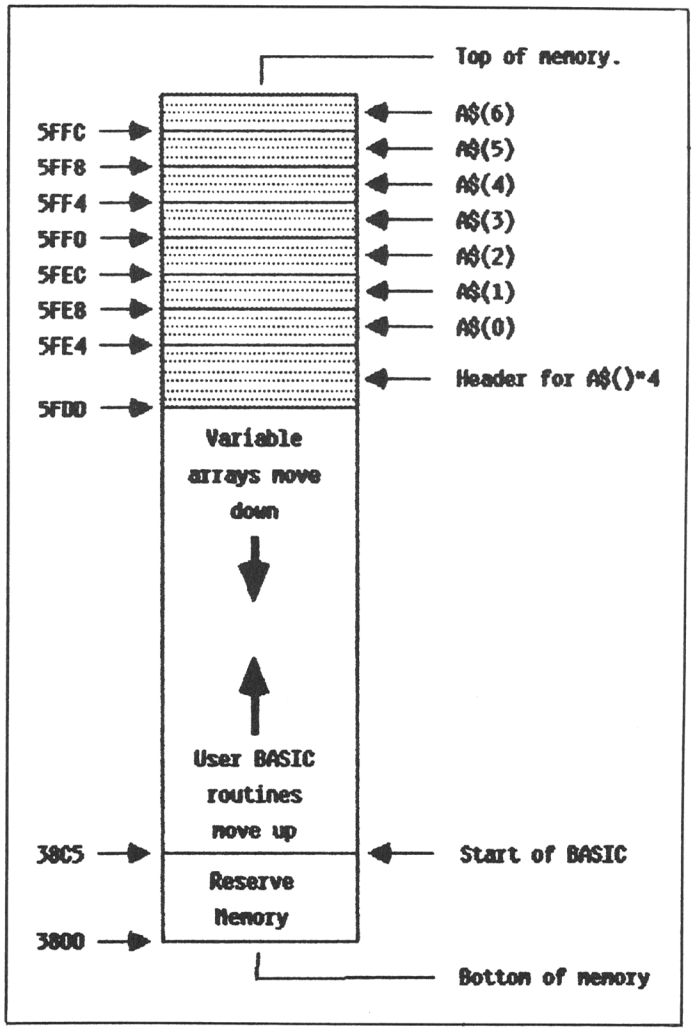
When the pocket computer is first turned on, the ROM program determines the bottom and top addresses of user RAM. The "page" (high order 8 bits) value of these locations are stored in the system RAM at addresses &7863 and &7864 respectively. This procedure is necessary as the range of user RAM can vary depending on which RAM expansion module (such as the CE-151, CE-155 or CE-161), if any, is installed in the PC. Thus, for example, if an 8K module was installed, location &7863 would contain &38 (which is the page portion of the address &3800 where RAM memory would begin). Location &7864 would contain &60. This is the page portion of the address &6000 which is one more than the top user RAM address (&5FFF) that is available in such a system.

(Knowing this information makes it possible to design a procedure that will automatically take account of the amount of memory in a user's system when it is used.)

When a BASIC programmer wants to create a variable array, it is necessary to issue a DIMension statement. This statement essentially blocks out space in user RAM for storage of the array elements. For purposes of illustration let us assume that the DIMension statement is the first statement contained in the user's BASIC program. Let us further assume that it is expressed as follows: DIM A$(6)*4. This means that a string array named A$() is being specified, that there are a total of 7 array elements (numbered 0 - 6) being reserved and that each element is to have room for 4 characters.

When the BASIC interpreter encounters this DIMension statement it will do the following: determine the top of memory (by examining system RAM location &7864), reserve 28 bytes of memory immediately below this location for storage of the array elements (7 elements with 4 characters reserved for each one), immediately below this it will record the array "header" information, (using 7 bytes of storage for this purpose). A summary of this process is shown in the accompanying diagram.

Diagram *Memory Map for Clearing an Array.*



The reason why it is necessary to assume that the DIMension statement is the first statement in the program in this description, is because any other arrays (or user-created variables) will be stored "beneath" this initial array. Thus, locating other arrays can become complicated and such complications are not needed at this point in the ML programming educational process. Right?

It will be worth knowing, so that you might customize such a routine to your own specific purposes, the following additional information about array elements: (1) The number of characters reserved for each element of a string array is precisely the number specified after the asterisk in the DIMension statement. If an asterisk is not used to specify such a value (which is limited to the range 1 to 80), then a value of 16 is assumed by the BASIC interpreter. (2) The number of characters reserved for each element of a numeric array is *always* eight! This is because all numeric values stored in such array elements are assumed to be in floating-point BCD format.

An Array-Clearing ML Routine

If we assume an array size of seven elements (numbered 0 through 6), with an element length of 4 characters, then we can calculate that the space needed by the array elements is exactly 28 bytes. (Note that this excludes the 7 bytes needed by the array "header." This is just as well, however, as we do not want to erase the contents of the array header!)

By examining the contents of memory location &7864, we can locate the start of memory. Actually, this location yields the first page address that lacks RAM, so user memory really begins on the highest possible address (&FF) on the next

Program *Routine for Clearing Memory that Terminates When Address Value Reached.*

PG	LC	B1	B2	B3	B4	B5	LABELS	MNEMONICS	COMMENTS
00	00	48	78				CLRMM4	LDXH #78	Set up register X to point to 16-bit address &78C0.
00	02	4A	C0					LDXL #C0	Set up register X to point to 16-bit address &78C0.
00	04	58	79					LDYH #79	Set up register Y to point to 16-bit address &7900.
00	06	5A	00					LDYL #00	Set up register Y to point to 16-bit address &7900.
00	08	B5	00				CLRM4	LDA #00	Load register A with zero.
00	0A	41						STAI (X)	Stuff zero into memory, advance memory pointer.
00	0B	94						LDA YH	Put contents of YH into the accumulator.
00	0C	86						CPA XH	Compare (YH-XH) to see if pointer page values are same.
00	0D	99	07					RBNZ CLRM4	Loop back if page values are not the same.
00	0F	14						LDA YL	Put contents of YL into the accumulator.
00	10	06						CPA XL	Compare (YL-XL) to see if pointer values are same.
00	11	99	0B					RBNZ CLRM4	Loop back if page values are not the same.

Program *Routine for Clearing Array Elements.*

PG	LC	B1	B2	B3	B4	B5	LABELS	MNEMONICS	COMMENTS
71	50	58	78				CARRAY	LDYH #78	Set up register Y to point to 16-bit address 67864.
71	52	5A	64					LDYL #64	This is where top of memory value stored by ROM routines.
71	54	15						LDA (Y)	Fetch top of memory value into accumulator.
71	55	DF						DEA	Decrement page value by one to point to next lower page.
71	56	08						STA XH	Store top of memory (adjusted) page value in register XH.
71	57	4A	FF					LDXL #FF	Store low portion of top of memory address in XL.
71	59	6A	1C				CARRA1	LDUL #1C	Set up counter (28 decimal here) in register UL.
71	5B	B5	00					LDA #00	Load the accumulator with zero.
71	5D	43					CARRA2	STAD (X)	Stuff accumulator into memory, then decrement pointer.
71	5E	62						DEUL	Decrement the counter value in UL (UL=UL-1).
71	5F	99	04					RBNZ CARRA2	If counter not zero, jump back to stuff another byte.
71	61	9A						RTS	Else, exit back to caller when counter equals zero.

lower page. Thus, if location &7864 contains the value &60 (indicating page &60 in the address &6000), the last location in RAM is at location &FF in the next lower page (&5F) or at address &5FFF. (This is precisely what would be encountered if a PC-1500 had an 8K RAM module such as the CE-155 installed.)

From there it is a simple matter to set up pointers and a byte counter within the CPU in order to erase the desired block of memory. However, now it will be appropriate to decrement the memory pointer as locations are cleared (instead of incrementing it as was done in previous routines). See the accompanying listing for the actual series of instructions that can accomplish this objective.

Check It Out With A Hybrid Program

You can test the operation of the array clearing routine by combining it with a BASIC program. The BASIC portion of the program can be used to DIMension the array, load the machine language routine into memory using POKE directives (since it is fairly short), and initialize the array with a set of known values.

Next, the initial values placed into the array can be displayed for checking purposes. The array-clearing ML routine can then be "called" using the CALL statement provided in BASIC. The contents of the array can then be displayed again to verify that the elements were properly cleared. This process may be repeated as long as desired for observational purposes. Here is the listing for such a *hybrid* program:

```
1000 "AA"DIM A$(6
     )*4
1010 GOSUB "CC"
1020 "BB"GOSUB "D
     D"
1030 GOSUB "EE"
1040 CALL &7150
1050 GOSUB "EE"
1060 GOTO "BB"
1090 END
1100 "CC"POKE &71
     50,&58,&78,&
     5A,&64,&15,&
     DF,8,&4A
1110 POKE &7158,&
     FF,&6A,&1C,&
     B5,0,&43,&62
     ,&99
1120 POKE &7160,4
     ,&9A
1130 RETURN
1200 "DD"FOR A=0
     TO 6
1210 A$(A)=STR$ (
     A)+STR$ (A)+
     STR$ (A)+
     STR$ (A)
1220 NEXT A
1230 RETURN
1500 "EE"WAIT 20
1510 FOR A=0TO 6
1520 PRINT A;"
      !";A$(A);"!
     ":PRINT " "
1530 NEXT A
1540 "FF"RETURN
```

(The nomenclature *hybrid* in this text refers to programs that combine BASIC and machine language methods within one general program, such as illustrated by this array-clearing example.)

Note that the ML routine is tucked into memory locations that are normally used for storing the string variables P$ and O$. Keep this in mind if you intend to meld this array-clearing capability into some other program!

INSIDE THE PC-1500

This is the sixth and concluding portion of the current series on the Sharp PC-1500. (Most of this information is also applicable to the Radio Shack PC-2.) *Norlin Rober, 407 North 1st Avenue, Marshalltown, IA 50158,* has been the provider of this extremely valuable information. We at *PCN*, on behalf of all our readers, would like to extend our thanks to him for a most informative treatise.

This final section picks up from where we left off in Issue 35, with a continuation of information regarding printing program lines....

(b) Subroutine D2EA may be used to locate the line whose number has been loaded into Register U. (A data byte, which will be added to the return address if the specified line does not exist, must follow the instruction calling this subroutine.) After execution of subroutine D2EA, load XH and XL with the contents of 78A6 and 78A7; decrement X twice; and execute subroutine D2D0. The specified line will be in the Input Buffer, and the SEARCH Pointer will Point to that line.

(c) The Program line that follows the one currently Pointed to by SEARCH may be Placed into the Input Buffer, with automatic uPdate of the SEARCH Pointer, by execution of subroutine D2B3.

(2) To Print the line stored in the Input Buffer:

(a) Execute subroutine B73F to store the line size into 79F5. (If the CSIZE set is larger than 2, it will be set to 2.)

(b) Subroutine AE3A will Perform a line feed with carriage return, then Print the line contained in the Input Buffer. Follow by subroutine A769 (Motor off)

MISCELLANEOUS (MECHANICAL)

(1) Paper Feed Key Disable/Enable

Disable: Subroutine A306
Enable: Subroutine A30A

(2) To move Paper (no uPdate of coordinates, GRAPH mode):

With n a two-byte signed binary number, subroutine AA0E feeds the Paper by n graPhic units. (Negative n will feed in reverse.) Precede by loading XH with 7B, XL with 7F, Y with n, and UL with 01.

(3) Pen UP/Down Control:

Pen UP: Subroutine AB08
Pen Down: Subroutine AAFA
Pen UP/Down: Subroutine AAE3 sets Pen uP if 79E9 contains zero, down if 79E9 contains FF

(4) To move Pen (no uPdate of coordinates, GRAPH mode):

(a) Move right, n graPhic units, with subroutine A28E. Precede by loading XL with n. Follow by subroutine A769 (Motor off)

(b) Move left, n graphic units, with subroutine A28B. Precede by loading XL with n. Follow by subroutine A769 (Motor off)

(c) Return pen to left end, without line feed, with subroutine A9D5. Follow by subroutine A769 (Motor off).

(5) To cycle pen to next color:

Execute subroutine A629. Follow by subroutine A769 (Motor off)

CASSETTE ROUTINES

TO SAVE A FILE ON CASSETTE

(1) Construct Header in output buffer area (7B60-7BAF)

(a) Execute subroutine BBD6 to form Lead-in, including file type code. Precede by loading A with file type code: 00=ML; 01=BASIC; 02=Reserve memory; 04=data file; others definable by user
(b) File name is optional; if to be included, its character codes must be stored into addresses 7B69-7B78 (maximum, 16 characters)
(c) Store beginning address of file being saved, into 7B82-7B83; store 1 less than the number of bytes in the file, into 7B84-7B85
(d) If a ML program (file type 00) is being recorded, store the beginning execution address (or default value FFFF) into 7B86-87

(2) Record Header onto cassette

(a) Store value of Cassette Parameter into 7879: 00=RMT 0, 10=RMT 1
(b) Execute Call B0 (BCE8)

(3) Record file

(a) Load X with beginning address (obtainable from 7B82-83); load U with 1 less than number of bytes (obtainable from 7B84-85)
(b) Execute Call AA (BD3C); terminate operation with Call B4 (BBF5)

TO LOAD A FILE FROM CASSETTE

(1) Construct Header in output buffer area, 7B60-7BAF

(a) Load A with code for file type sought
(b) Execute subroutine BBD6
(c) Store (optional) file name into 7B69-78

(2) Find file on cassette

(a) Store value of Cassette Parameter into 7879: 80=RMT 0, 90=RMT 1
(b) Execute CALL B0 (BCE8). (A search will begin for a previously-recorded Header having the file type specified. If found, its file name is displayed and stored into 7B91-A0. If this name differs from the one specified, the search continues)
(c) If flag C is set, there was a checksum error; end with Call B4

(3) Load file into computer

(a) Load X with the address into which the file is to be loaded, and load U with 1 less than the number of bytes to be loaded. (The values for these parameters that were originally recorded are now obtainable, respectively, from 7BAA-AB and 7BAC-AD)
(b) Execute Call AA (BD3C). Following loading, Register X will contain the address immediately following the last address loaded
(c) If flag C is set, there was a checksum error; end with Call B4

(4) Terminate operation of the recorder with Call B4 (BBF5)

CONTROL OF RELAYS

Subroutine BF28 may be used to operate the remote control relays. The effect of this subroutine is determined by the contents of A.

A=03: Close relay, REM 0
A=05: Open relay, REM 0

A=09: Close relay, REM 1
A=11: Open relay, REM 1

RECORD, LOAD, OR VERIFY BLOCK OF DATA

Register X must be loaded with the beginning address of the block of memory involved, and Register U must contain 1 less than the number of bytes in that block. Then Call AA (BD3C) will perform the operation specified by the contents of the Cassette Parameter (7879), as follows:

When 7879 contains 00, the block of data is recorded onto tape.

When 7879 contains 80, the data read from tape is transferred to memory. If a checksum error occurs, there will be a return, with flag C set.

When 7879 contains 40, data read from tape is compared to the data in the block of memory, without any transfer of data. If a discrepancy is found, there is a return, with flags C and V set; if a checksum error occurs, a return with flags C and H set.

In each of the above cases, Call B4 (BBF5) may be used to terminate the operation; it will open the relay, shutting off the recorder.

RECORD OR LOAD A SINGLE BYTE

Subroutine BDCC will record the byte contained in the accumulator.

Call A4 (BDF0) will read one byte from tape, and load it into the accumulator. This routine also checks whether the BREAK key has been used; if it has, flag C is set.

1984 Index - Ordered By Type of Article

Article	Author	Issue	Page
Announcement: "The Commuter" Portable	PCN, Staff	31	1
Announcement: Casio FX-750P	PCN, Staff	35	1
Announcement: Casio PB-700	PCN, Staff	33	1
Announcement: Flight Computer Uses PC-1500	PCN, Staff	32	8
Announcement: Graphics Package for PC-2	PCN, Staff	32	8
Announcement: Hewlett-Packard HP-110 Portable	PCN, Staff	34	1
Announcement: Hewlett-Packard HP-71B	PCN, Staff	32	1
Announcement: Sharp PC-1350	PCN, Staff	36	1
Commentary: Watch Pocket	PCN, Staff	35	15
Commentary: Watch Pocket	PCN, Staff	36	16
Program: Addressing Pixels (PC-1500/PC-2)	Pollet, Patrick L.	33	9
Program: Alarm Clock (PC-1250/51/60/61 & PC-1211/PC-1)	Auersbacher, Emerich	36	3
Program: Alternate Character Sets (PC-1500/PC-2)	Rober, Norlin	35	10
Program: Appointment Calendar (PC-1500/PC-2)	Eichman, Steve	32	4
Program: Binary Magic (PC-1500/PC-2)	Gibson, John R.	33	11
Program: Formatted Mortgage (PC-1500/PC-2)	Hartmann, Peter V.	32	6
Program: Improved Vertical Lister (PC-1500/PC-2)	Rober, Norlin	34	8
Program: Isometric Drawings (PC-1500/PC-2)	Chrystie, Richard H.	34	10
Program: Linear Systems (PC-1500/PC-2)	Rober, Norlin	34	8
Program: Little Editor (PC-1500/PC-2)	Jackson, H. David	31	5
Program: Perspective Plot (PC-1500/PC-2)	Rober, Norlin	35	7
Program: Pocket Wordpro (PC-1500/PC-2)	Rober, Norlin	35	4
Program: Schedule Plotter (PC-1500/PC-2)	Chrystie, Richard H.	32	2
Program: Shell Game (PC-1500/PC-2)	Gibson, John R.	33	10
Program: Weather Forecaster (HP-71B)	PCN, Staff (Conversion)	36	5
Review: Casio PB-700	Woodhead, Jeffrey	36	2
Review: CE-501A, CE-501B, CE-505A Sharp ROM Modules	Sutliff, Robert	33	5
Review: Ce-502A, -502B, -504A, -507A Sharp ROM Modules	Sutliff, Robert	34	3
Review: Sharp PC-1261	PCN, Staff	35	2
Review: The Hewlett-Packard HP-71B	PCN, Staff	33	3
Review: The Sharp PC-5000 Portable	PCN, Staff	31	2
Technical Info: Clearing Memory and Arrays (PC-1500/PC-2)	PCN, Staff	36	7
Technical Info: Inside the PC-1500 (Part 1)	Rober, Norlin	31	11
Technical Info: Inside the PC-1500 (Part 2)	Rober, Norlin	32	9
Technical Info: Inside the PC-1500 (Part 3)	Rober, Norlin	33	12
Technical Info: Inside the PC-1500 (Part 4)	Rober, Norlin	34	12
Technical Info: Inside the PC-1500 (Part 5)	Rober, Norlin	35	12
Technical Info: Inside the PC-1500 (Part 6)	Rober, Norlin	36	12
Technical Info: Updates on Inside the PC-1500	Rober, Norlin	36	15

NORLIN UPDATES

Norlin Rober has noted a few errors in the ongoing *Inside the PC-1500* series that has been appearing in 1984 issues of *PCN*. Please make the following corrections:

Issue 31 - page 12 - line 785D should read: If 80, Katakana displayed; 00, displayed and . . . etc.

Addresses 786C to 786F ***are*** used by ROM, to temporarily save the contents of 787C-7F while the Display Buffer is temporarily saved in the String and Output Buffers.

Issue 31 - page 13 - 788E should read: TRACE parameter: 00, **ending** execution of **previous** line; 01, starting execution of **new line**; . . . etc.

Issue 31 - page 14 - location 79DA: . . . is obtained from 79DB - DC.

Issue 32 - page 11: In the middle of the page, the Output Buffer addresses for storage of tape header information are incorrectly given as beginning with 7A . . . the 7A should be replaced by **7B**.

Issue 32 - page 15: Under DECIMAL-TO-BINARY CONVERSION, the memory address of CALL D0 is incorrectly stated as D0F9; it should be **D5F9**.

Note too, that system RAM addresses that are "unused" by the PC-1500 actually **are** used by the CE-158. These are **7850 to 7858 and 79FA to 79FE**.

Norlin also notes that there is a mistake in his article in Issue 34 on page 8. The example given is not correct and its use can cause readers to think that the program is not operating correctly. The example printed should be changed so that the

constant term in the third equation should be 3 (not 1) and the constant in the fourth equation should be 4 (not 3). Additionally, when used with an *old* PC-1500 (having early versions of ROM), the list of coefficients will not be LPRINTed correctly. To remedy this situation, change the last part of line 32 to read:

```
LPRINT "C  ";A(I,N+1): . . .
```

instead of:

```
LPRINT "C  ";A(I,J): . . .
```

Finally, Norlin has a few general tips to pass on the PC-1500 users:

1. A BASIC program inadvertently cleared by NEW can be recovered as follows:

(1) POKE &7750, 204, 101, 202, 105, 73, 0, 69, 221, 153, 4, 70, 202, 103, 154

(2) CALL &7750

Note: If the number of the first line was larger than 255, it will be changed by this procedure. It may be corrected with the usual editing procedures. You can use the above routine, in some cases, to recover a program that has been lost by a crash. Key ALL RESET and execute NEW 0 *before* using this routine for crash recovery!

2. It is possible to make a BASIC program halt with automatic display of the RESERVE template (for whichever of groups I, II or III is currently set). This technique adds a nice touch to a menu-driven program. The program instructions for doing it are simply: POKE &7880,0:CALL &CB8B

3. Here is a way to clear RAM completely. Every byte of RAM will be cleared to zero, including system RAM, after which the power-up routine will automatically be executed.

(1) POKE &4000, 40, 36, 8, 10, 170, 255, 253, 186, 211, 197

(2) CALL &4000

(3) Key CL and execute NEW 0

Finally, although the instruction manual does not mention it, the BASIC statement ON ERROR GOTO 0 will cancel an ON ERROR directive!

Available Only by Prepaid Subscription for a Calendar Year Period (January - December). You are sent back issues for the calendar year to which you subscribe, at the time you enroll.

__ I am interested. Please send more information. I have the following Model Pocket Computer: _____

__ Enroll me as a 1985 Subscriber (Issue numbers 37-44). $24.00 in U.S. (U.S. $30.00 to Canada/Mexico. Elsewhere U.S. $40.00 payable in U.S. funds against a U.S. bank.)

__ Enroll me as a 1984-85 Subscriber (Issue numbers 31-44). $42.00 in U.S. (U.S. $51.00 to Canada/Mexico. Elsewhere U.S. $70.00 payable in U.S. funds against a U.S. bank.)

__ Enroll me as a 1983-85 Subscriber (Issue numbers 21-44). $78.00 in U.S. (U.S. $93.00 to Canada/Mexico. Elsewhere U.S. $120.00 payable in U.S. funds against a U.S. bank.)

__ Enroll me as a 1982-85 Subscriber (Issue numbers 11-44). $102.00 in U.S. ($125.00 to Canada/Mexico. Elsewhere U.S. $160.00 payable in U.S. funds against a U.S. bank.)

__ Check here if paying by MasterCard or VISA. Please give credit card information below.

Orders must be accompanied by payment in full. All checks must be magnetically encoded, payable in U.S. funds and drawn against a U.S. bank.

Name: _____

Addr: _____

City: _____ State: ___ Zip: _____

MC/VISA #: _____

Signature: _____ Exp. Date: _____

mail this order form to:
POCKET COMPUTER NEWSLETTER
P.O. BOX 232, SEYMOUR, CT 06483

POCKET COMPUTER NEWSLETTER

P.O. Box 232, Seymour, CT 06483

FROM THE WATCH POCKET

As we close-out our fourth year of publishing this unique newsletter, it is interesting to note that Sharp Electronics continues to spearhead the effort to produce good quality, low cost PCs. Of course, many of the other pocket computer pioneers are still hanging in there. Casio continues to put up a good fight. HP is doing OK. Even Panasonic continues to push its HHC, though primarily to specialized markets.

There are probably well over a million PCs in use in the United States today. That is only about one for every 250 persons. It took about 20 years for calculators to reach their present ubiquitous status. Chances are it won't take as long for PCs to reach the same level of distribution. Based on that projection, the growth of the industry should be steep during the coming years. *PCN* looks forward to covering this advancing, exciting field.

By the way, during 1985 we will increase the frequency of publication to eight times (instead of the current six issues-per-year). However, rising postal and publishing costs dictate that we revert back to a smaller type style and our previous 8-page format. The combination of steps being taken will yield continued quality coverage of the pocket computing field in a timely manner. We all thank you for your continued support and extend our best wishes for pocket computing happiness during 1985, our fifth straight year of publication!